**IT 114 - ADVANCED PROGRAMMING FOR INFORMATION TECHNOLOGY**

**Case Studies**

**1. Formulating the Problem**

**1.1 Problem Description**

We must create a program that We must create a program that enables two users to chat. Using GUIs implement one user as the server and the other as the client. The sever has two text areas: one for entering text and the other (noneditable) for displaying text received from the client. The client has two text areas: one for receiving text from the server and the other for entering text. When the user presses the enter key, the current line is sent to the server or client.

**1.2 Verbalization**

*What is the goal?*

Create a program that lets the enables two users to chat

*What are the givens?*   Two separate GUIs talking to each other

*What are the unknowns?*   How they will communicate

1.3 **Information Elicitation**

*Goal*

Create a program that – lets the server and client talk to each other --- using two separate GUIs --- deletes a specified index from the linked list --- gives a popup for error messages --- gives a popup for search results.

*Givens*   The amount of connections for the program, 2 (client and server)

*Unknowns*   what kind of data will be passed along to eachother

*Conditions*   The user must connect to the same port of the server

**2. Planning the Solution**

**2.1 Solution Strategy**

Create two GUIs that have a 2 text areas. One for data input and one for data output. Create 2 classes. One for the server and its GUI, and one for the client and its GUI. Then create the server and socket on the server side with a port for the client to connect to. Create a socket for the client to that the client can connect to the server. Take the outputstream from the client and send it to the inputstream for the server and vice versa. Print the text into the text areas.

**)**

**2.2 Goal Decomposition**

*Sub-goal 1*

Create server GUI

*Sub-goal 2*

Create client GUI

*Sub-goal 3*

Create server and port

*Sub-goal 4*

Create socket for server

*Sub-goal 5*

Create client socket

*Sub-goal 6*

Take send outputstream from client to server inputstream and vice versa

*Sub-goal 7*

Print the texts in the input textAreas of the client and servers

**2.3 Resources**

*Relevant Information*

**Port:** 8000

**3. Designing the Solution**

**)3.1 Structure Chart**

*First Level Decomposition*



*Goal Refinement*

**Sub-goal 1**

Create a class called Server that will hold the server and its GUI

**Sub-goal 2**

Create a class called client that will be used by the client to send messages to the server through its own GUI

**Sub-goal 3**

Create 2 text Areas in the GUI for input and output

**Sub-goal 4**

If the client sends a message, print the message in its own text area and in the text area of the server.

**Sub-goal 5**

If the server sends a message, print the message in its own text area and in the text area of the client.

*Second Level Decomposition*



The second level decomposition shows operations between the Client and the Server users. The process starts with the Client sending a message to the server, and connecting to the server. Once that connection has been established, the message is displayed on the server and client TextAreas displaying what the Client user sent. The lambda expression in the Client side picks up what was written and appends it to the TextArea of the Client. So they can see what they wrote like in an actual chat messenger. The Client and Server can now speak to each other and send each other messages. When the server sends a message, its picked up by the lambda expression and printed to the TextArea, the message is also sent through the output stream and received by the client socket. The message is then displayed on the TextArea of the client.

**3.2 Module and Data Specifications**

**Name**: Input – Client user sends message to Server

**Input**: String text

**Output**: Connection is established and message is displayed in the TextArea of the client and Server

**Logic**: The server waits for a connection request to its port. Once a request is sent through the initial message from the client, the server accepts it and a connection is established. The message is then read by the inputStream of the Server and displayed in the textArea, the message is also displayed in the TextArea of the Client.

**Name**: Input – Server sends message to Client

**Input**: String text

**Output**: message sent from server to client is displayed on TextArea of both client and server.

**Logic**: The message is sent via the socket but received through the OutputStream, it is then then read by the inputStream of the Client and displayed in the textArea, the message is also displayed in the TextArea of the Server.

**3.3 Logic**

*Logic*

1. Display empty text area for client
2. Client sends message to Server
3. Server Establishes connection

3.1 Client OutputStream sends message via the Client Socket

3.1 Server InputStream receives message via the Server socket

3.2 BufferedReader takes the data from the InputStream and prints it

In the text area of the Server.

3.3 Message also appears in the client side text area

4.0 Server sends message to Client

4.1 Message is sent via the Server OutputStream and the Server

socket.

4.2 The Client socket receives the OutputStream of the Server and

relays it to the InputStream of the Client

4.3 Buffered reader takes the data from the InputStream and prints

It in the text area of the client

4.4 Message also appears in the text area of the Server

*Algorithm Description*

The process of doing this is that the client first sends a message to the Server. When the client sends a message to the server, it first asks to establish a connection, then the server accepts that connection. After the connection has been established the message is received by the Inputstream of the Server via the socket and uses Buffered Reader to print the data into the Text Area. When the server sends a message, it is picked up by the Server Output Stream and the socket sends the message to the Client Socket. The Client Socket sends transmits that message to the Client Input Stream. Using the Buffered Reader the message is printed into the Text Area of the Client.

**4. Translation**

**4.1** **Source Code**

I removed the indents to try to make it easier to read in this document. With the indents it was hard to look at.

//======================================================   
// Name : Emad Tirmizi  
// SID : 31400222  
// Course : IT114   
// Section :   
// Instructor : Maura Deek  
// T.A :   
//======================================================   
//======================================================   
// Assignment # : 5  
// Date : 11/30/2018  
//======================================================   
//======================================================   
// Description: We must create a program that gives  
// enables two users to chat. Using GUIs implement one   
// user as the server and the other as the client. The   
// sever has two text areas: one for entering text and  
// the other (noneditable) for displaying text received  
// from the client. The client has two text areas: one  
// for receiving text from the server and the other for  
// entering text. When the user presses the enter key,  
// the current line is sent to the server or client.  
  
//======================================================   
  
**import** java.io.**\***;  
**import** java.net.**\***;  
**import** java.util.Date;  
**import** javafx.application.Application;  
**import** javafx.application.Platform;  
**import** javafx.scene.Scene;  
**import** javafx.scene.control.ScrollPane;  
**import** javafx.scene.control.TextArea;  
**import** javafx.scene.control.TextField;  
**import** javafx.stage.Stage;  
**import** javafx.scene.layout.GridPane;  
**import** javafx.geometry.Insets;  
**import** javafx.scene.layout.ColumnConstraints;  
**import** javafx.geometry.Orientation;  
**import** javafx.geometry.Pos;  
**import** javafx.geometry.VPos;  
**import** javafx.event.EventHandler;  
**import** javafx.scene.input.KeyEvent;  
**import** java.io.InputStream;  
  
**public** **class** Server **extends** Application{  
  
Stage window;  
Stage scene;  
  
**public** **static** **void** main(String[] args) {  
launch(args);  
//Client.main(new String[0]);   
}  
  
  
  
**@**Override  
**public** **void** start(Stage primaryStage) **throws** Exception{  
window **=** primaryStage;  
window.setTitle(**"Server"**);  
  
//I used grid the get the layout I wanted  
GridPane grid **=** **new** GridPane();  
//The padding is to essentially set the margins for which my nodes won't go past  
grid.setPadding(**new** Insets(10,10,10,10));  
grid.setVgap(10);  
grid.setHgap(15);  
  
//TextArea where the chat will be displayed  
TextArea ts **=** **new** TextArea();  
ts.setMaxWidth(500);  
//set editable to false so people dont write in the wrong text Area  
ts.setEditable(**false**);  
GridPane.setConstraints(ts, 0, 0);  
  
//text field to enter input  
TextField tw **=** **new** TextField();  
tw.setMaxWidth(500);  
GridPane.setConstraints(tw, 0, 1);  
  
//add them to grid layout  
grid.getChildren().addAll(ts, tw);  
  
//set grid layout to the layout of the scene and create scene  
Scene scene **=** **new** Scene(grid, 500, 200);  
window.setScene(scene);  
//show the Stage  
window.show();  
  
//Lambda expression so when data is input it will show up in the User's window aswell  
tw.setOnAction(e **->** {  
String ItoClient **=** **""**;  
ItoClient **=** tw.getText();  
//append the text to the Text Area  
ts.appendText(**"Server: "** **+** ItoClient **+** **'\n'**);  
//clear text area so user doesn't have to do it manually  
tw.clear();  
});  
  
//enables GUI to be loaded while waiting for connection request from Client  
**new** Thread(() **->** {  
  
**try**{  
//create server with at port 8000  
ServerSocket serverSocket **=** **new** ServerSocket(8000);  
//wait for request then accept  
Socket socket **=** serverSocket.accept();  
//take input from client outputstream received via socket  
InputStream fromClient **=** socket.getInputStream();  
//output from server to client inputStream sent via socket  
OutputStream toClient **=** socket.getOutputStream();

**while** (**true**) {  
//InputStream Reader to read from Input Stream  
InputStreamReader inputReader **=** **new** InputStreamReader(fromClient);  
//Buffered Reader to read it from StreamReader  
BufferedReader br **=** **new** BufferedReader(inputReader);  
//Create String to hold the text  
String line **=** br.readLine();  
//Append text to Text area so the Server can read the text from the client  
ts.appendText(**"Client: "** **+** line **+** **'\n'**);  
}

}  
  
**catch**(IOException ex){  
ex.printStackTrace();  
}  
});  
}  
}  
  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**import** java.io.**\***;  
**import** java.net.**\***;  
**import** java.util.Date;  
**import** javafx.application.Application;  
**import** javafx.application.Platform;  
**import** javafx.scene.Scene;  
**import** javafx.scene.control.ScrollPane;  
**import** javafx.scene.control.TextArea;  
**import** javafx.scene.control.TextField;  
**import** javafx.stage.Stage;  
**import** javafx.scene.layout.GridPane;  
**import** javafx.geometry.Insets;  
**import** javafx.scene.layout.ColumnConstraints;  
**import** javafx.geometry.Orientation;  
**import** javafx.geometry.Pos;  
**import** javafx.geometry.VPos;  
**import** javafx.event.EventHandler;  
**import** javafx.scene.input.KeyEvent;  
**import** java.io.InputStream;  
  
**public** **class** Client **extends** Application{  
Stage window;  
Stage scene;  
  
//Initialize the GUI  
**public** **static** **void** main(String[] args) {  
launch(args);   
}  
  
**@**Override  
**public** **void** start(Stage primaryStage) **throws** Exception{  
window **=** primaryStage;  
window.setTitle(**"Client"**);  
  
//I used grid the get the layout I wanted  
GridPane grid **=** **new** GridPane();  
//The padding is to essentially set the margins for which my nodes won't go past  
grid.setPadding(**new** Insets(10,10,10,10));  
grid.setVgap(10);  
grid.setHgap(15);  
  
//TextArea where the chat will be displayed  
TextArea tc **=** **new** TextArea();  
tc.setMaxWidth(500);  
//set editable to false so people dont write in the wrong text Area  
tc.setEditable(**false**);  
GridPane.setConstraints(tc, 0, 0);  
  
//text field to enter input  
TextField tw **=** **new** TextField();  
tw.setMaxWidth(500);  
GridPane.setConstraints(tw, 0, 1);  
  
//add them to grid layout  
grid.getChildren().addAll(tc, tw);  
  
//set grid layout to the layout of the scene and create scene  
Scene scene **=** **new** Scene(grid, 500, 200);  
window.setScene(scene);  
//show the Stage  
window.show();  
  
//InputStream set to null  
InputStream fromServer **=** **null**;  
//OutputStream set to null  
OutputStream toServer **=** **null**;   
  
//Lambda expression so when data is input it will show up in the User's window aswell  
tw.setOnAction(e **->** {  
String ItoServer **=** **""**;  
ItoServer **=** tw.getText();  
//append the text to the Text Area  
tc.appendText(**"Client: "** **+** ItoServer **+** **'\n'**);  
//clear text area so user doesn't have to do it manually  
tw.clear();  
});  
  
**try** {  
//send request to establish connection at Localhost port 8000  
Socket socket **=** **new** Socket(**"localhost"**, 8000);  
//output from server to client inputStream sent via socket  
toServer **=** socket.getOutputStream();  
//take input from client outputstream received via socket  
fromServer **=** socket.getInputStream();  
  
//InputStream Reader to read from Input Stream  
InputStreamReader inputReader **=** **new** InputStreamReader(fromServer);  
//Buffered Reader to read it from StreamReader  
BufferedReader br **=** **new** BufferedReader(inputReader);  
//Create String to hold the text  
String line **=** br.readLine();  
//Append text to Text area so the Server can read the text from the client  
tc.appendText(**"Server: "** **+** line **+** **'\n'**);  
}  
**catch**(IOException ex){  
tc.appendText(ex.toString() **+** **'\n'**);  
}  
}  
}

**)4.2 Program and Module Description**

Main

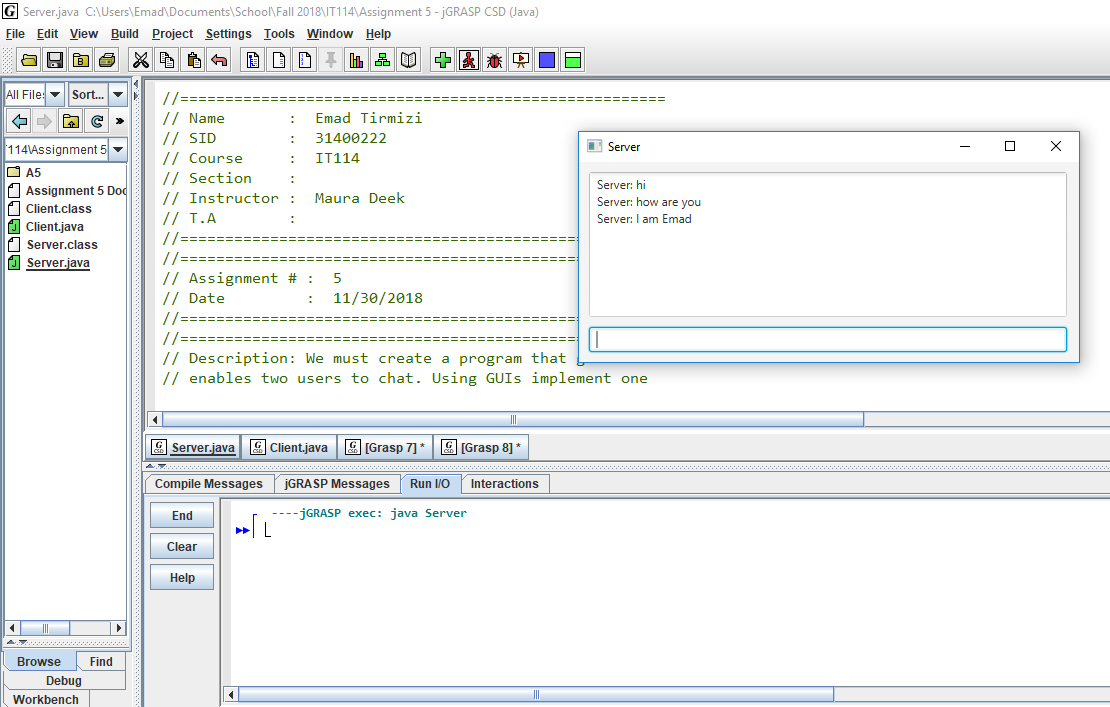
The main function creates the program GUI when started.

**5. Solution Testing**

Test the program with following data domain:

Any text, I don’t know if the Client and Server actually speak to each other. I didn’t know how to launch both programs simultaneously on JGRASP. I’ve spent a week just trying to figure that out. I tried adding Client.main(new String[0]) in server to launch Client when I launch Server but it didn’t work.

Server GUI showing Server input Text



Client GUI showing Client input Text

